Ngày 23 tháng 5 năm 2023

SM2023 Day 01

A. MONSTERS Solution

Xây dựng đồ thị vô hướng với tập đỉnh:

$$V = \{(i,j): 1 \le i \le m, 1 \le j \le n, a[i,j] \neq' \#'\}$$

Mỗi đỉnh $(i,,j) \in V$ sẽ kề tối đa với 4 đỉnh theo 4 hướng tới các ô chung cạnh

Thuật toán được thực hiện theo 2 bước:

+Bước 1:) Từ tất cả các đỉnh 'M' tiến hành duyệt đồ thị ưu tiên chiều rộng (BFS) tính được mảng $dM[i,j]$= Khoảng cách ngắn nhất từ một ô 'M' tới ô (i,j)

+ Bước 2:) Từ ô 'A' thực hiện BFS một lần nữa. Tuy nhiên điều kiện kề từ ô $(i,j) \to (u,v)$ thêm rằng buộc $dA[i,j] + 1 < dM[u,v]$ với $dA[i,j]$ là khoảng cách ngắn nhất từ ô 'A' tới ô (i,j). Trong bước này mỗi khi gặp được một đỉnh biên (i,j) ($i = 1$ hoặc $i = m$ hoặc $j = 1$ hoặc $j = n$) thì dừng và tìm ngược lại đường đi từ ô này đến ô 'A' (đay là đường đi cần tìm).

Tham khảo phần code cho bước 2:

```
void BFSA() {
    if (bien(Axp.first, Axp.second)) {
        Akt = Axp;
        return;
    }
    for(int i = 1; i <= m; ++i)
        for(int j = 1; j <= n; ++j)
            dA[i][j] = INT_MAX;
    L = 1, R = 0;
    dA[Axp.first][Axp.second] = 0;
    q[++R] = Axp;
    while (L <= R) {
        II node = q[L++];
        int i = node.first, j = node.second;
        for(int k = 1; k <= 4; ++k) {
            int ii = i + dh[k], jj = j + dc[k];
            if (ii >= 1 && ii <= m && jj >= 1 && jj <= n && a[ii][jj] != '#') {
                if (dA[ii][jj] > dA[i][j] + 1 && dA[i][j] + 1 < dM[ii][jj]) {
                    dA[ii][jj] = dA[i][j] + 1;
                    pre[ii][jj] = k;
                    q[++R] = {ii, jj};
                    if (bien(ii, jj)) {
                        Akt = {ii, jj};
                        return;
                    }
                }
            }
        }
    }
    Akt = {0, 0};
}
```

B. CIRCUS Solution

Xây dựng đồ thị với tập đỉnh $V = \{(i, j, k, c): 1 \le i \le m, 1 \le j \le n, 1 \le k \le 4, 1 \le c \le 5\}$
Thể hiện diễn viên đang ở ô (i, j) quay mặt theo hướng k và màu của sector chạm đất là c. Tất nhiên $a[i, j] \neq' \#'$

Từ một ô (i, j, k, c) có thể có 3 ô kề:

1. Tiến lên đến ô $(i1, j1, k1, c1)$ với $i1 = i + dr[k], j1 = j + dc[k], k1 = k, c1 = 1 + (c + 3)\%5$
2. Quay phải: $i1 = i, j1 = j, k1 = 1 + (k + 2)\%4, c1 = c$
3. Quay trái: $i1 = i, j1 = j, k1 = 1 + k\%4, C1 = c$

Bài toán quay về tìm đường đi ngắn nhất từ đỉnh $s = (i_s, j_s, 2, c_s)$ đến tập đỉnh:

$$T = \{(i_T, j_T, c_T, k): k = 1,2,3,4\}$$

Tham khảo code chính:

```
void BFS() {
    L = 1;
    R = 0;

    q[++R] = IV (II (Cs, 2), II (is, js));
    dis[Cs][2][is][js] = 0;
    color[Cs][2][is][js] = 1;

    while (L <= R) {
        IV u = q[L++];
        int i = u.sc.ft, j = u.sc.sc, c = u.ft.ft, k = u.ft.sc;

        II v = ke (u, k);
        if (v != II (0, 0)) {
            int i1 = v.ft, j1 = v.sc, c1 = 1 + (c + 3) % 5;
            if (color[c1][k][i1][j1] == 0) {
                color[c1][k][i1][j1] = 1;
                q[++R] = IV (II (c1, k), II (i1, j1));
                dis[c1][k][i1][j1] = dis[c][k][i][j] + 1;
            }
        }

        int k1 = 1 + k % 4;
        if (color[c][k1][i][j] == 0) {
            color[c][k1][i][j] = 1;
            dis[c][k1][i][j] = dis[c][k][i][j] + 1;
            q[++R] = IV (II (c, k1), II (i, j));
        }

        k1 = 1 + (k + 2) % 4;
        if (color[c][k1][i][j] == 0) {
            color[c][k1][i][j] = 1;
            dis[c][k1][i][j] = dis[c][k][i][j] + 1;
            q[++R] = IV (II (c, k1), II (i, j));
        }
    }
```

```
    int ans = 2e9;
    for (int i = 1; i <= 4; ++i)
        if (dis[Ct][i][it][jt] != 0)
            ans = min (ans, dis[Ct][i][it][jt]);
    if (ans == 2e9)
        cout << -1;
    else
        cout << ans;
}
```

C. MINPATH Solution

Đây là bài tập cơ bản về thuật toán Dijkstra. Chú ý rằng một hệ quả quan trọng của thuật toán tìm đường đi ngắn nhất là ta có "đồ thị đường đi ngắn nhất" với đỉnh u kề đỉnh v khi và chỉ khi $d[v] = d[u] + L(u, v)$.

Do $L(u, v) > 0$ nên đồ thị trên là đồ thị có hướng không có chu trình-DAG. Trên đồ thị này sẽ có một thứ tự topo:

$$tp_1, tp_2, \dots, tp_n$$

Đây là thứ tự lấy ra các đỉnh khỏi hàng đợi ưu tiên trong Dijkstra (các cung của "đồ thị đường đi ngắn nhất" hướng từ trái qua phải theo thứ tự trên)

Theo thứ tự trên ta có 3 bài toán qui hoạch động:

+1) : Đếm số đường đi ngắn nhất từ đỉnh 1 đến đỉnh u. Đặt $f[u]$ là số đường đi cần tìm ta có:

$$f[u] = \sum_{d[u]=d[v]+L(v,u)} f[v], f[1] = 1$$

+2): Đặt $g[u]$ là số cạnh ít nhất trên đường đi ngắn nhất từ 1 đến u ta có:

$$g[u] = \min\{g[v] + 1 : d[v] + L(v, u) = d[u]\}$$

+3): Đặt $h[u]$ là số cạnh nhiều nhất trên đường đi ngắn nhất từ 1 đến u ta có:

$$h[u] = \max\{h[v] + 1 : d[v] + L(v, u) = d[u]\}$$

Tham khảo các đoạn code tính ba bài toán qui hoạch động trên. Chú ý $tp[1..n]$ là thứ tự lấy ra các đỉnh khi thực hiện thuật toán Dijkstra:

```
    for(int u = 1; u <= n; ++u)
        cnt[u] = 0;
    for(int i = 1; i <= m; ++i) {
        int u = tp[i];
        if (u == 1)
            cnt[u] = 1;
        for(auto e : adj[u]) {
            int v = e.first, L = e.second;
            if (d[v] == d[u] + L)
                cnt[v] = (cnt[v] + cnt[u]) % MOD;
        }
    }
    cout << cnt[n] << ' ';

    for(int u = 1; u <= n; ++u)
        MinNode[u] = n + 1;
    for(int i = 1; i <= m; ++i) {
        int u = tp[i];
        if (u == 1)
            MinNode[u] = 0;
        for(auto e : adj[u]) {
            int v = e.first, L = e.second;
```

```
            if (d[v] == d[u] + L)
                MinNode[v] = min(MinNode[v], MinNode[u] + 1);
        }
    }
    cout << MinNode[n] << ' ';

    for(int u = 1; u <= n; ++u)
        MaxNode[u] = -1;
    for(int i = 1; i <= m; ++i) {
        int u = tp[i];
        if (u == 1)
            MaxNode[u] = 0;
        for(auto e : adj[u]) {
            int v = e.first, L = e.second;
            if (d[v] == d[u] + L)
                MaxNode[v] = max(MaxNode[v], MaxNode[u] + 1);
        }
    }
    cout << MaxNode[n] << '\n';
```

D. HIGHSCORE Solution:

Xây dựng đồ thị với tập đỉnh $V = \{1,2, \dots, n\}$. Các cạnh là các đường hầm với trọng số $-x$. Khi đó bài toán qui về tìm đường đi ngắn nhất từ đỉnh 1 đến đỉnh n (đáp số sẽ là $-d[n]$). Do trọng số đồ thị có thể âm nên ở đây phải sử dụng thuật toán Ford Bellman để tìm đường đi ngắn nhất. Chú ý rằng ngay cả trường hợp đồ thị có chu trình âm thì vẫn có đường đi ngắn nhất từ 1 đến n không qua chu trình âm này. Thuật toán được tiến hành theo 2 bước:

+B1:) Từ đỉnh n thực hiện BFS trên đồ thị ngược để đánh dấu tất cả các đỉnh có thể đến được n trên đồ thị xuôi.

+B2:) Tiến hành Ford Bellman trên đồ thị với hạn chế là chỉ xét các đỉnh được đánh dấu từ B1.

```
long long Ford_Bellman() {
    if (nho[1] == 0)
        return -1LL;
    for(int i = 1; i <= n; ++i)
        d[i] = INF;
    queue<int> q;
    for(int i = 1; i <= n; ++i)
        color[i] = cnt[i] = 0;
    d[1] = 0;
    q.push(1);
    color[1] = 1;
    ++cnt[1];
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        color[u] = 0;
        for(auto e : adj[u]) {
            int v = e.first, L = e.second;
            if (nho[v] && d[v] > d[u] + L) {
                d[v] = d[u] + L;
                if (color[v] == 0) {
                    color[v] = 1;
                    q.push(v);
                    ++cnt[v];
```

```
                    if (cnt[v] > 2 * n)
                        return -1LL;
                }
            }
        }
    }
    return (d[n] != INF) ? -d[n] : -1LL;
}
```

E. VACATION Solution:

Bài toán qui về thực hiện Q truy vấn, mỗi truy vấn sẽ tìm đường đi ngắn nhất từ đỉnh u đến đỉnh v. Chú ý rằng đồ thị có thêm điều kiện là nếu $(u, v) \in E$ thì u hoặc v là một đỉnh trong số k đỉnh được chọn.

Do vậy có thể thực hiện thuật toán như sau:

+B1:) Thực hiện k lần Dijkstra từ các đỉnh được chọn ta có mảng d[i,u] = khoảng cách từ đỉnh thứ i được chọn đến đỉnh u.

+B2:) Thực hiện k lần Dijkstra từ các đỉnh được chọn nhưng trên đồ thị ngược ta được $f[i, u]$ là khoảng cách từ đỉnh u đến đỉnh được chọn thứ i.

+B3:) Khi đó với mỗi truy vấn $(u\ v)$ ta có đáp số là $\min\{f[u, i] + d[i, v]: i = 1,2, \dots, k\}$

Có thể cải tiến thuật toán trên một chút khi trả lời các truy vấn là:

- Nếu u là đỉnh thứ i được chọn thì kết quả bằng $d[i, v]$
- Nếu nếu u không là đỉnh được chọn thì kết quả là $\max\{L(u, j) + d[rank_j, v]\}$ với j là đỉnh kề với đỉnh u còn $rank_j$ là thứ tự của j trong số các đỉnh được chọn.